SEVENSTAX-SNMP

User Manual

Revision No.:	1.7
State:	Release
Author:	sevenstax GmbH

Initial version:	31/01/08
Last change:	24/01/11
Last Review:	07/12/10
Publication:	Public
Filename:	sevenstaxSNMP_UserManual_v01_07

Copyright (c) 2011 by SEVENSTAX GmbH
This document is an intellectual property of sevenstax GmbH. Unauthorized copying and distribution is prohibited.

Table of Contents

1 Abstract....5

2 Product Definition....6

2.1 Features....6

2.2 Modules....7

2.3 Requirements....7

2.4 Compatibility....8

2.4.1 SNMP Standards....8

2.4.2 SNMP Commands....8

2.4.3 SMI Data Types....8

2.4.4 Restrictions....9

3 How it works....11

3.1 Overview on SNMP Processing....11

3.2 Initialization....11

3.3 Generating Traps....12

3.3.1 Steps to use Traps....12

3.3.2 Processing Traps....12

3.3.3 Trap usage notes....12

3.4 Debug Support....13

4 Creating a MIB Module....14

4.1 MIB base node number....14

4.2 Design your own MIB....15

4.3 Convert the MIB into Software....15

4.3.1 Basic software classification....15

4.3.2 Handling of Object-Identifiers (OIDs and Sub-OIDs)....16

4.3.3 Creating a GroupTable....17

4.3.4 Creating an ObjectTable....17

4.3.5 Optimized usage and reduced ROM size....18

4.4 Implement Object-CallbackHandlers....18

4.5 Handling of SNMP-Tables (Dynamic Objects)....18

4.5.1 Overview....18

4.5.2 Processing tables *19*
4.5.3 Example 'udpTable' *19*
4.6 Register the MIB **20**
5 SNMP Module **21**
5.1 Abstract **21**
5.2 Data Types **21**
5.3 SNMP Functions **21**
5.3.1 stxSNMP_Init (fnApplCallback, fnMIBHandler) *21*
5.3.2 stxSNMP_Tick ( ) *22*
5.3.3 stxSNMP_AgentSetup(usPort, szCommRd, szCommRdWr) *22*
5.3.4 stxSNMP_AgentStart ( ) *23*
5.3.5 stxSNMP_AgentStop ( ) *23*
5.3.6 stxSNMP_TrapAgentSetup (DestIP, usTrapPort, szTrapComm) *23*
5.3.7 stxSNMP_TrapAgentStart ( ) *24*
5.3.8 stxSNMP_TrapAgentStop ( ) *24*
5.3.9 stxSNMP_TrapAgentSend (aulTrapOID) *25*
5.4 Adjustable SNMP Parameters **26**
5.4.1 SNMP Protocol Parameters *26*
5.4.2 SNMP Buffer Sizes *26*
6 MIB Module **27**
6.1 Abstract **27**
6.2 Internal Handling of OIDs **27**
6.3 Static and Dynamic Objects **28**
6.3.1 Static Objects and Tables *28*
6.3.2 Dynamic Objects and Tables *28*
6.4 Data Types **29**
6.4.1 MIBs, GroupTables and ObjectTables *29*
6.4.2 MIB_GROUP_TYPE *29*
6.4.3 MIB_OBJECT_STAT_TYPE *30*
6.4.4 MIB_OBJECT_DYN_TYPE *31*
6.5 MIB Functions **32**
6.5.1 stxMIB_Init ( ) *32*
6.5.2 stxMIB_Tick ( ) *32*
6.5.3 stxMIB_RegisterModule(ulMIBBaseOID, ucMIBType, pGroupTable, szMIBName) *33*
6.5.4 Callback MIB_CBFCT_GROUP (ucTableNr) *33*
6.5.5 Callback MIB_CBFCT_OBJSTAT (Cmd,Var,ppdata,SetObjtT,VarObjT,Rights,pLen) *34*

6.5.6 Callback MIB_CBFCT_OBJDYN (Cmd,Var,ppdata,RetIdx,SetObjtT,VarObjT,Rights,ObjTIdx,pulOID,pLen)....35

6.6 Adjustable MIB Parameters....36

7 RFC1213-MIB....37

7.1 Abstract....37

7.2 Using the RFC1213-MIB....37

7.3 Public Functions....37

7.3.1 stxMIB_1213_Init (fctApplCallback)....38

7.3.2 stxMIB_1213_Tick ( )....38

7.3.3 Callback MIB1213_CBFCT_APPL (ucVarID, ppData, pDataLen)....38

7.4 Adjustable Parameters....39

8 SEVENSTAX-MIB....40

8.1 Abstract....40

8.2 Using the SEVENSTAX-MIB....40

8.3 Generating Traps....41

8.4 MIB Structure....41

8.4.1 stxTraps....41

8.4.2 stxTrapManager....41

8.4.3 stxTriggers....42

8.4.4 stxVariables....42

8.5 Public Functions....42

8.5.1 stxOID_Init ( )....42

8.5.2 stxOID_TrapInit ( )....43

8.5.3 stxOID_Tick ( )....43

8.6 Adjustable Parameters....43

9 Tools....45

9.1 MIB-Validators....45

9.2 MIB Code-Generators....45

9.3 MIB Browsers....45

9.4 SNMP Simulator....45

10 Appendix A: SEVENSTAX-MIB....46

11 Change History....53

1 Abstract

SNMP – **S**imple **N**etwork **M**anagement **P**rotocol - is a protocol used to exchange management information via IP/UDP between network devices.

SNMP uses MIBs – **M**anagement **I**nformation **B**ase – as a fundamental data structure with dedicated access and description methods.

SEVENSTAX-SNMP is part of the SEVENSTAX Internet Protocols Suite and specially designed for small embedded devices. It is a collection of software modules working as a SNMPv2 Agent. It can be used to receive SNMP-Requests, to send SNMP-Responses and also supports sending of SNMP-Traps.

This document gives the user an overview on how

- SEVENSTAX-SNMP works
- customer specific MIBs can be added
- customer specific OIDs are supported
- customer specific Traps can be stimulated

To help the user understanding the SEVENSTAX-SNMP product, two independent example implementations are given, which are free to be used and/or extended by the customer:

- MIB-1213 – containing common network settings and information
- SEVENSTAX-MIB – containing several usage showcases

Both are integrated into a ready to uses reference application.

For basic information on how to use the SEVENSTAX Internet Protocols Suite, please refer to the SEVENSTAX-TCP/IP Users Manual.

2 Product Definition

2.1 Features

SEVENSTAX-SNMP supplies the following features:

SNMP Support:

- Acting as SNMPv2 Agent
- Reception of SNMPv2 SET, GET, GET-NEXT, GET-BULK request messages
- Transmission of SNMPv2 SET, GET response and TRAP messages
- Adjustable access rights via SNMPv2 Communities Names
- optimized code for very fast response time (e.g. <1 ms at 50 MHz)
- SNMP-Object types OID, INT32, OCTSTR, NULL, SEQOF, SEQ, IPADDR, CNT32, GAUGE32, TTICKS.
- SNMP Error-Index and Error-Status
- SNMP statistics

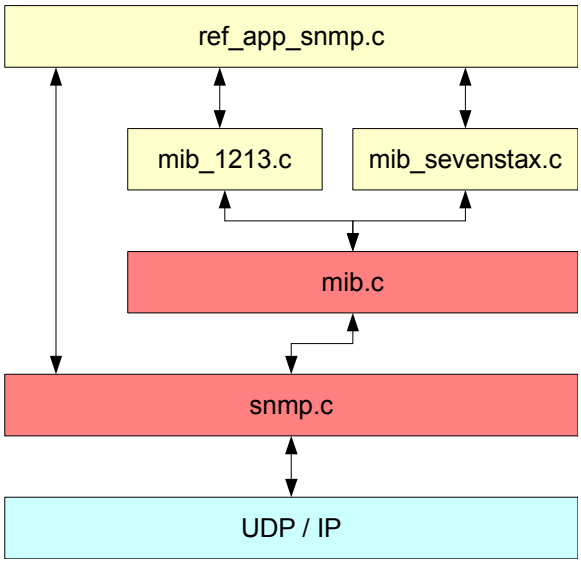
MIB Support:

- adjustable number of different MIB modules
- easy to add user defined MIB modules
- Static (direct) and dynamic (tables) MIB elements supported
- optimized code and OID lookup table for fast response time

Development Support:

- ready-to-use and extendible Network and System data MIB (MIB-1213)
- example MIB demonstrating any kind of access (SEVENSTAX-MIB)
- easy GET / SET access to application data via callback functions
- a SNMPv2-Trap-Agent API separated and easy to use

2.2 Modules



SEVENSTAX-SNMP consist of these modules (based on the UDP/IP stack):

- ref_app_snmp.c a sample application acting as a SNMP Agent incl. Traps
- mib_sevenstax.c a sample SEVENSTAX-MIB with sevenstax specific information
- mib_1213.c a sample RFC1213-MIB for network and system information
- mib.c the MIB core processing module
- snmp.c the SNMP core protocol module

For details to these module, please refer to following chapters.

2.3 Requirements

To ease the use of this manual and SEVENSTAX-SNMP,

the **Developer** should have knowledge about and/or experience with

- basic SNMP protocol behaviour
- design of a MIB and MIB-Objects (e.g. SMI, ASN.1, BER)
- SEVENSTAX TCP/IP product

The **Target System** should be provided with

- SEVENSTAX TCP/IP
- SEVENSTAX-SNMP
- two free UDP-Port for SNMP-Messages and SNMP-Trap-Messages

2.4 Compatibility

2.4.1 SNMP Standards

The SNMP-Protocol and the handling of Managed-Information-Data is described a bunch of RFCs and ISO-Documents. Some of them are replaced by newer ones, some are extended by other ones. Three basic SNMP-Standards are defined in RFCs, SNMP-version 1, 2 and 3. It's not the purpose of this document to describe the differences of the versions. But to provide the user with the most important advantages of SNMP on very small embedded devices, SEVENSTAX-SNMP bases on SNMPv2c, which seems to be the most popular version on embedded devices.

SEVENSTAX-SNMP bases on SNMPv2, this is an overview of the relevant RFCs:

- ISO X.680 - Abstract Syntax Notation One (ASN.1)
- ISO X.690 - Specification of Basic Encoding Rules (BER)
- RFC 1155 - Structure and Identification of Management Information
- RFC 1156 - Management Information Base for Network Management
- RFC 1157 - A Simple Network Management Protocol (SNMP)
- RFC 1212 - Concise MIB Definitions
- RFC 1213 - Management Information Base for Network Management MIB-II
- RFC 1215 - A Convention for Defining Traps for use with the SNMP
- RFC 1901 - Introduction to Community-based SNMPv2
- RFC 1905 - Version 2 of the Simple Network Management Protocol (SNMPv2)
- RFC 2578 - Structure of Management Information Version 2 (SMIv2)

2.4.2 SNMP Commands

SEVENSTAX-SNMP is acting as an Agent only. Therefore the following list of supported commands results. This is a list of supported received SNMP Commands of RFC1905:

- get-request
- get-next-request
- get-bulk-request
- set-request

This is a list of supported transmitted SNMP Commands of RFC1905:

- response
- snmpV2-trap

Please see RFC1905 for details of these commands.

2.4.3 SMI Data Types

This is a list of supported primitive or derived data types of RFC1155:

- INTEGER
- OCTET STRING
- NULL
- OBJECT IDENTIFIER

- SEQUENCE
- SEQUENCE OF
- IpAddress
- Counter
- Gauge
- TimeTicks

Please see RFC1155 for details of these types.

2.4.4 Restrictions

As all SEVENSTAX Internet Suite products, SEVENSTAX-SNMP is a compromise between maximized conformity and minimized resource requirements. It is designed for use in very small embedded systems, so most of the following points should not be regarded as an indispensable requirement:

#	***Restriction***	***Note***
1.	SNMP-Version <= V2c supported only	Currently, features of SNMPv1 and V2 are supported only. SNMPv3 includes an extended security concept using encryption standards, which commonly exceed the resource of small embedded devices.
2.	Base-MIB-Area “Management” and “Enterprise” supported only	Company specific MIBs are sub-part of the MIB number base “.1.3.6.1.4.1. - Enterprise”. Most RFC based MIBs are sub-part of base “.1.3.6.1.2 – Management”. Both are supported.
3.	Two global Communities supported only	Community-based View per Object is not supported. But Object-Access-Rights distinguish between “read-only” and “read-write” for each individual object. The Access-Rights are mapped to two adequate global Community names only, (default: “public” and “private”) which can be setup at runtime.
4.	GET-Request with 1 Variable-Binding only	GET-Requests are supported for only 1 Variable-Binding, any further is ignored. Some SNMP-Managers are able to request a complete line of a table, which results in n x Variable-Bindings in one Request. This is currently not supported. If required, please send a sequence of GET-Requests for the table entries.
5.	SNMP-Table variation at runtime not supported	The variation of a tables and table cells (create/delete) is not supported.
6.	SNMP-Table SET-Request not supported	Currently GET/GET-NEXT Requests into parts of SNMP-Tables are supported only.
7.	Data type 'Opaque' not supported	The RFC1155 defined data type to pass arbitrary syntax 'Opaque' is currently not supported.

If required, please contact SEVENSTAX for an extension of SNMP or MIB functionality.

3 How it works

3.1 Overview on SNMP Processing

This is an overview, on how SNMP-Messages are processed inside SEVENSTAX-SNMP:

1. Application specific MIBs are created, converted into source code and are registered to the MIB-Module at runtime. SNMP-UDP-Ports are opened and the network is connected.
2. A MIB-Browser Application (acting as a SNMP-Manager) sends a SNMP-Request to the embedded system (acting as an SNMP-Agent).
3. The SNMP-Module validates the received UDP-SNMP frame (e.g. Port and Community) and extracts the Request-Type (e.g. GET) and Object-Identifier (OID).
4. The Request-Type and OID are given the MIB-Module, which now starts searching a tree-search inside each registered MIB:
    - First the basic OID ciphers are checked: If it's not part of '**iso.org.dod.internet**' nor '**iso.org.dod.internet.private.enterprise**', the request is rejected.
    - Then the list of **GroupTables** of the MIB is checked
    - If it's found, the list of **GroupTable-Entries** of this GroupTable gets checked.
    - If at any of the above times the compared OID is higher than the requested OID, the search is stopped and the next registered MIB gets scanned.
    - If the OID is not found in any MIB, the MIB-Module returns the adequate error code
5. The complete OID is accepted, then the specific **OID-CallbackHandler** gets called. This is an application and OID-specific function which returns valid data and pointer to it – or an error code, if necessray.
6. The MIB-Module returns the OID data (pointer, size, type, errrocode) back the SNMP-Module.
7. The SNMP-Module constructs an adequate SNMP-Response based on the returned OID data and sends it out to the network.

Details on how MIBs are processed and specific Object data is requested is demonstrated inside software modules for RFC1213-MIB and SEVENSTAX-MIB.

SNMP messages using the the wrong version, port, Msg-Type, Request-ID or Community are silently discarded and do not result in SNMP-Error messages. Unavailable OIDs, wrong data types or access rights are responded with the adequate error number.

To prevent access conflicts and reduce management, only one simultaneous SNMP-Manager Request is supported. SNMP-Requests are responded in typical time of < 1ms, so the probability of an unanswered request minimized.

3.2 Initialization

The following steps have to be considered to initialize SNMP with MIB:

- Activate *#define SNMP_SUPPORTED = 1* in 'features.h'. This enables SNMP and MIB module (of course UDP_SUPPORTED must be enabled too).

- Assure *#define UDP_MULTI >= 2* in 'features.h', to additionally support two SNMP UDP ports at runtime.
- If you like to use or extend the prepared RFC1213-MIB please enable, *#define MIB1213_SUPPORTED = 1* in 'features.h'.
- If you like to use the prepared SEVENSTAX-MIB please enable, *#define STXOID_SUPPORTED = 1* in 'features.h'.
- Call *stxSNMP_Init()* once to set-up SEVENSTAX-SNMP to register the applications callback function for receiving a SNMP message, and to connect the SNMP with the MIB-Module.
- Call *stxMIB_Init()* once to set-up MIB software module.
- In your main loop or operating system task call *stxSNMP_Tick()* and *stxMIB_Tick()* repeatedly, to keep it alive and check for incoming SNMP messages.

3.3 Generating Traps

SEVENSTAX-SNMP internally distinguishes two kinds of SNMP-Agents:

- a Standard SNMP-Agent, which responds SNMP-Requests, and
- a Trap-Agent, which is able to initiate a SNMPv2-Trap triggered by the application.

Please note, that the Trap-Agent has to be activated and set up by a separated API.

3.3.1 Steps to use Traps

These are the essential steps to send a Trap:

1. Setup the Trap-Agent with essential parameters of the remote Trap-Manager with *stxSNMP_TrapAgentSetup()*: Destination-IP-Address, Destination-UDP-Port-Nr. (typical 162) and the SNMP-CommunityName. These parameters are application or Trap-manager specific.
2. Open a free local UDP-port for usage as Trap-manager Port by calling *stxSNMP_TrapAgentStart ().*
3. Shoot out a single SNMP-Trap at any time you need to, by calling *stxSNMP_TrapAgentSend()*. Note, that the given parameter must be defined inside one of the local registered MIBs!

3.3.2 Processing Traps

If the Trap-Send function is called, internally the following steps are executed:

- check, whether the Trap-Agent is activated and well configured
- scan the registered MIBs for the given Object-ID
- construct a valid SNMPv2-Trap-Message, including the local system time as
- send out the Trap using the predefined Trap-Port, IP and CommunityName.

These steps are done synchronously inside the call of *stxSNMP_TrapAgentSend()*, to assure rapid transmission.

3.3.3 Trap usage notes

As for all UDP-based protocols, there is no transport-layer guarantee, that the Trap really has been sent.

Furthermore, there is no handshaking mechanism inside the SNMP-Protocol, which supports any kind of acknowledge for a received Trap. If needed, this has to be handled by a specific application behaviour between Agent and Manager (e.g. timeout-resending of traps, if no following GET-request received from Manager).

Please note, that SEVENSTAX-SNMP supports only one user-defined Object for a Trap.

3.4 Debug Support

As all SEVENSTAX products, a printf() interface is supported to deliver any kind of info, warning error. This might be usable at development process only and enhances test phase speed.

Please enable (#define as 1) one/all of the following switches in 'features.h':

- DEBUG_STX – main switch to enable DebugOuts
- SNMP_DEBUG – enables DebugOuts of the SNMP Core module
- MIB_DEBUG – enables DebugOuts of the MIB Core module
- MIB_DETAILS_DEBUG – enables more detailed DebugOuts for MIB
- SNMPAGENT_DEBUG – enables DebugOuts of the Reference Application
- MIB1213_DEBUG – enables DebugOuts of the RFC1213-MIB module
- STXOID_DEBUG – enables DebugOuts of the SEVENSTAX-MIB module
- STXOID_TRAPDEBUG – enables Trap details of SEVENSTAX-MIB

If not enabled, no code is generated and ROM resources are saved.

Please note, that some of the #defines (especially the MIB-Search and Traps) might produce debugouts at a high data rate.

4 Creating a MIB Module

MIBs are designed in a standard format, using "***S**tructured **M**anagement **I**nformation*" - **SMI** - format in text files, using "***A**bstract **S**yntax **N**otation **1***" - **ASN.1** and "***B**asic **E**ncoding **R**ules"* - **BER**.

All of these standards are defined in RFCs or ISO documents. It's not part of this manual to describe, how SMI, ASN.1 or BER is to be used. Please follow appropriate public literature and manuals and examples, to understand it.

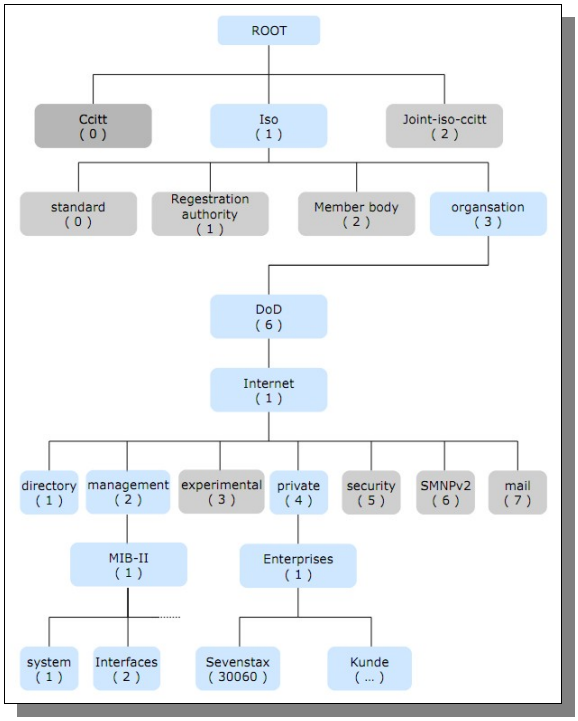
These steps have to be done to create and use your own application specific MIB:

1. Decide the base MIB node number (Management or Enterprise)
2. Design your MIB in SMI form (using ASN.1 and BER conform syntax)
3. Convert the MIB-Objects into software constant arrays
4. Implement object specific callback handlers
5. Register you MIB at application runtime

Please follow the next chapters for details. Please note, that some Restrictions apply to a MIB supported by SEVENSTAX-SNMP, which should be known before creating it.

4.1 MIB base node number

The following graphic is an reduced overview of the standard MIB-Tree:


ROOT
Ccitt (0)
Iso (1)
Joint-iso-ccitt (2)
standard (0)
Regestration authority (1)
Member body (2)
organsation (3)
DoD (6)
Internet (1)
directory (1)
management (2)
experimental (3)
private (4)
security (5)
SMNPv2 (6)
mail (7)
MIB-II (1)
Enterprises (1)
system (1)
Interfaces (2)
Sevenstax (30060)
Kunde (...)

The first step to design your own MIB is decide the number area, in which your MIB should be integrated. Two base areas are supported by SEVENSTAX-MIB:

- '**iso.org.dod.internet.management**' which is '**.1.3.6.1.2**' or

- **iso.org.dod.internet.private.enterprise**' which is **'.1.3.6.1.4.1'.**

A large amount of MIB use cases is already covered by RFC-MIBs, which are based on one of those base number areas above. We recommend to explore them, before completely design your own MIB. In this case, your MIB can be based on '**iso.org.dod.internet.management** '. The MIB text file already exists and is well formatted. You can go on with the next step “Convert the MIB into Software“.

If you are sure, that you need a company or application specific MIB, your node number base is **iso.org.dod.internet.private.enterprise**'. The following number is the IANA registered number of your company (e.g. SEVENSTAX registered '30060').

All subsequent numbers (nodes and OIDs) in software base on these node number '**.1.3.6.1.2'** or **'.1.3.6.1.4.1.<IANA>'**, which helps to reduce memory consumption and speeds up the tree-search.

4.2 Design your own MIB

It's not purpose of this document to describe how MIB files are created. Please follow appropriate public literature and manuals and examples, to understand creating MIB files. But for a first glance, please find the SEVENSTAX-MIB as an example at the appendix part of this manual.

Note: We strongly recommended to validate your MIB file, before importing it into tools like Wireshark or MIB-Browsers. Please see chapter 9.1 MIB-Validators for details.

4.3 Convert the MIB into Software

SEVENSTAX-SNMP needs a kind of lookup table, to find the requested Node-OID or Object-ID inside it's registered MIBs. The conversion from a ASN.1-notation MIB-File into sourcecode has to be done manually.

But SEVENSTAX spent some effort to keep this conversion as easy as possible. Please see the following chapters for details.

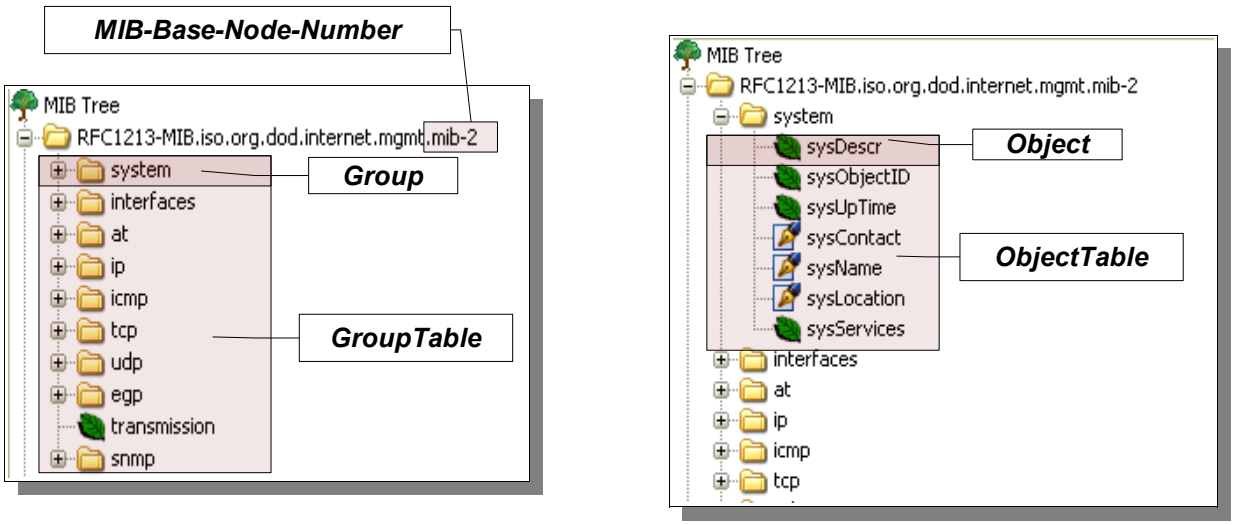
4.3.1 Basic software classification

To understand, how MIBs are designed in SEVENSTAX-SNMP software, the basic principles and required structure of the SEVENSTAX-SNMP for one MIB is as follows:

1. All Object-IDs and Node-IDs are relative to the ***MIB-Base-Node-Number***
2. Any MIB consists of ***Groups***, which are are hold in a ***GroupTable***.
3. Any Group consists of ***Objects***, which are hold in one ***ObjectTable***.
4. Any Object references to exactly one ***ObjectHandler***.

This rule applies to all registered MIBs and to all Nodes and Objects, independently of the OID number depth.

As an example, the following pictures shows the structure of the prepared RFC1213-MIB:



1. This MIB bases on 'iso.org.dod.internet.management' which is '.1.3.6.1.2'.
2. The ***MIB-Base-Node-Number*** in this case is '1', which is for 'mib-2'.
3. The MIB consist of a ***GroupTable***, which contains ***Groups*** from '*system*' until '*snmp*'.
4. The ***Group*** '*system*' consist of a ***ObjectTable***, which contains the ***Objects*** from 'sysDescr' until 'sysServices'.

All essential elements are described in C-Types:

- One MIB refers to a table of Groups of type MIB_GROUP_TYPE (see chapter 6.4.2)
- One Group refers to a table of Objects of type MIB_OBJECT_STAT_TYPE or MIB_OBJECT_DYN_TYPE (for static or dynamic Objects).
- One Object refers to a Callback of the MIB_CBFCT_OBJSTAT or (for static or dynamic Objects).

Please follow the references above to view C-Code examples and see chapter 6.3 for the difference on dynamic and static objects.

4.3.2 Handling of Object-Identifiers (OIDs and Sub-OIDs)

One point to ease creation and handling and to reduce resource requirements of registered MIBs, is the SEVENSTAX-internal use of so called "***Sub-Object-identifier***" - ***Sub-OIDs***. This applies to Group- and ObjectTables in the same way. The main features of Sub-OIDs are:

- Each table entry (Group- or Object-Table) contains a Sub-OID.
- A Sub-OID is always relative to Base-Node-Number of the overlying element: A Group-Sub-OID is relative to the MIB-Base-Node-Number, and a Object-Sub-OID is relative to the Group-Sub-OID. This speeds up the MIB-Search process and reduces ROM resources.
- A Sub-OID may be of a variable length. This allows to integrate a dedicated Sub-node to the table, without the need to include all Sub-OIDs between these Groups.
- All Sub-OIDs inside a table are in a ascending order (essential!) But: There is no need of a consecutive order, number gaps are allowed. This prevents from adding unnecessary elements to a table.

Please refer to chapter 6.2 Internal Handling of OIDs for details.

4.3.3 Creating a GroupTable

A GroupTable is the basic element of a MIB in SEVENSTAX-SNMP. The following properties apply to a GroupTable:

- the data type is an array of entries of type MIB_GROUP_TYPE (see chapter 6.4.2 for details).
- The Group-Node-Identifiers are designed as described above in chapter 4.3.2.
- Every entry contains a reference to a CallbackHandler of type MIB_CBFCT_GROUP, which is able to return the static address of the ObjectTable behind that Group.
- Every entry contains a Type identifier, that distinguishes between static and dynamic objects of the ObjectTable behind that group

This is an example code of a MIB containing several GroupTable entries:

```
/* 1213RFC-MIB Standard Table Group '.1.3.6.1.2.1' */
STATIC_stx MIB_GROUP_TYPE MIB_1213_GroupTable[] = {
    /* size-of-entry, oid-sub-part, function-to-get-table, type dyn/stat */
    { 1, {1,0},    stxMIB1213_GetGroup, MIB_TABLE_STAT },      // '.1.3.6.1.2.1.1'   system
    { 1, {2,0},    stxMIB1213_GetGroup, MIB_TABLE_STAT },      // '.1.3.6.1.2.1.2'   interfaces
    { 2, {2,2,0},  stxMIB1213_GetGroup, MIB_TABLE_DYN  },      // '.1.3.6.1.2.1.2.2'   interfaces table
    ...
    { 1, {11,0},   stxMIB1213_GetGroup, MIB_TABLE_STAT },      // '.1.3.6.1.2.1.11'    snmp
    MIB_GROUP_LAST
};
```

In this example, the function *stxMIB1213_GetGroup()* gets implicitly called, whenever the MIB searches for an object residing inside that table. E.g., if the ObjectID '.1.3.6.1.2.1.1'. It returns the reference to the GroupTable '*MIB_1213_Group_System*' . This reference will later on be used to search for the OID (and its handler function) inside this Group.

4.3.4 Creating an ObjectTable

The ObjectTable is the basic element of a Group in SEVENSTAX-SNMP. There two kinds of ObjectTables – static and dynamic. The following properties apply to a ObjectTables:

- the data type is an array of entries of type MIB_OBJECT_STAT_TYPE or MIB_OBJECT_DYN_TYPE (please chapter 6.4.3 or 6.4.4 for details).
- The Object-identifiers are designed as described above in chapter 4.3.2.
- Every entry contains a reference to a CallbackHandler of type MIB_CBFCT_OBJSTAT or Callback MIB_CBFCT_OBJDYN, which must be able to support a GET or SET Request for that Object-Value.
- Every entry contains a unique VariableIdentifier. It might be used to identify the Object in a more easy and application specific way, e.g. if only one single CallbackHandler has to support several Objects.
- Every entry contains a AccessRight-Identifier, that distinguishes between the access rights 'read-only' and 'read-write' for that object.

Please see MIB_OBJECT_STAT_TYPE and MIB_OBJECT_DYN_TYPE and for examples on ObjectTables.

4.3.5 Optimized usage and reduced ROM size

The 2-Step-rule (Groups/Objects) mentioned above, applies to the complete MIB. This means, that

- one Group always references to exactly one ObjectTable,
- a Group never directly references an Object.
- an Object never references to another Object, Group or Table
- each Group- and Object-Table entry contains a Sub-OID (remaining rest) of the complete OID only.

It looks like there is a problem to describe Objects with a depth > 3, because no Sub-Groups are allowed. But inside the ObjectTable the user is free to use different Sub-OID lengths! So the definition of further Sub-Groups can flexible be done inside the ObjectTable.

And the user is free to omit an Object OID (therefore generating gaps inside the Object-Table). This means, that there is no need to completely define a continuous sequence of all available OIDs. This advantage can be used in ObjectTables and GroupTables.

These both features are supported to flexible support any kind of MIB structure, with regard on reduced ROM resources and to speed up the MIB search algorithm.

4.4 Implement Object-CallbackHandlers

SEVENSTAX-SNMP does know on which Object is requested, but on how! This has to be described by the application software. The interface between the MIB and the application are so CallbackHandlers, which have to be implemented by the user.

There are two kind of Objects available:

- ***StaticObjects***, which OID is already known at compilation time, and
- ***DynamicObjects***, which OID is defined and constructed at runtime (see 4.5)

Both are supported by SEVENSTAX-SNMP, each by a dedicated predefined C-Function prototype. (see chapter 6.5.5 and 6.5.6). The common property of both is that they receive

- a simple Sub-Object-ID
- the SNMP-Request method (e.g. GET or SET)
- a pointer to the source buffer (if SET-Request)
- a reference to a pointer and datasize buffer to return the result

NOTE: The CallbackHandler is free to accept or reject the request - the adequate SNMP-Response will be generated automatically. But, if the CallbackHandler accepts and returns application data for the Object, it is responsible to return a valid static pointer and data size value!

4.5 Handling of SNMP-Tables (Dynamic Objects)

4.5.1 Overview

Multi-dimension SNMP-Tables need a special handling. It's not mission of this manual to describe, how SNMP-Tables are described or addressed. But some main properties have to regarded:

- Tables might be n-dimensional (e.g ARP-Table 1 dimension)
- Tables have dynamic structure, line/row numbers might change
- Tables need at least one Index-value, which is used to build up the dynamic OID.
- A Table-Index might be a complex data type (e.g. IP address)
- Table cells can not directly be accessed with GET,
- Table cells can be accessed with a sequence of GET-NEXT only

4.5.2 Processing tables

To support this behaviour, a dedicated ObjectdataType “MIB_OBJECT_DYN_TYPE” and a special CallbackHandler “Callback MIB_CBFCT_OBJDYN” prototype is defined. The function itself has to be user implemented. The main challenge of this CallbackHandler is to

- extract the current table position (OID, input value)
- calculate the next table position (OID, return value)

To help the user application to detect the current position, SEVENSTAX-SNMP extracts the current table index-value from the requested OID (parameter “*pulOID*” of Callback MIB_CBFCT_OBJDYN):

- if it is the first GET-NEXT request (starting at Table-Base-OID), “*pulOID*” contains “.0”
- if it is a subsequent GET-NEXT request , “*pulOID*” contains a Sub-OID sequence in a format exactly defined by the index-values of this table.

The CallbackHandler must be able temporarily generate an local OID based on the table index values. Then the CallbackHandler can compare the delivered OID with the local copy and go one scanning its own table, until the next-bigger OID is generated. This OID and the adequate value are returned back as the 'next table position' to the MIB module.

If no bigger local OID is generated, the CallbackHandler has to return '**SNMP_ERR_OID_END',** to indicate, that the table is completely delivered.

As source code examples, you might use one of RFC1213-MIB function for the UDP-Table, TCP-Table or ARP-Table, which all use different optimized methods for generating the GET-NEXT OID.

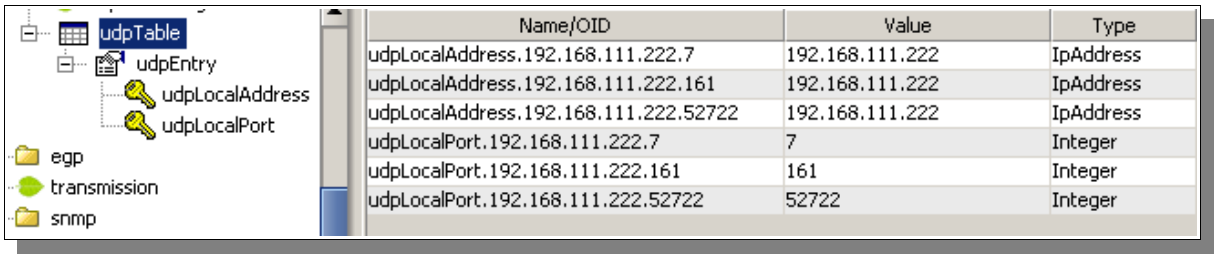
4.5.3 Example 'udpTable'

E.g. for the 'udpTable' inside RFC1213-MIB. The 'udpTable' has 2 dimensions:

- a dynamic number of 'udpEntry's
- a pair of 'udpLocalAddress' and 'udpLocalPort' (per udpEntry)

The responding OID always has the same structure, the Table Base-OID extended with the two index values: **.1.3.6.1.2.1.7.5.1.<udpLocalAddress>.<udpLocalPort>**. Because the requested value is part of the responded OID, the requested OID occurs twice – in the OID and in Value field.

As an example, the following picture shows the result of a GET-NEXT through the UDP-Table:



For the first GET-NEXT request (Table-Start at '.1.3.6.1.2.1.7.5.1'), the delivered *pulOID* references a single '.0', which means, we have to start with delivering the first OID.

For subsequent GET-NEXT requests, the delivered *pulOID* always references the last OID delivered to the SNMP-Manager. The Manager uses this old value to request the GET-NEXT value. This OID can be compared with a local copy of the dynamic UDP-Table, to distinguish the next Table Index and OID.

The last GET-NEXT requests for OIDs of this udpTable is indicated, when no further udpTable Entries are available. This has to be notified by the CallbackHandler to the MIB-Module by the return value **'SNMP_ERR_OID_END'**. This stimulates the MIB-Module to continue searching in the GroupTable, which will simply result in delivering the NEXT OID found ever found inside the registered MIBs. The SNMP-Manager accepts this OID (completely outside the udpTable) and detects, that no further UDP-Table Entries are available, and therefore stops sending further GET-NEXT Requests for this table.

4.6 Register the MIB

SEVENSTAX-SNMP supports a definable number of MIBs, which can be setup at compilation time (see 6.6). To save a prepared MIB, follow these steps:

- define the ***MIB-Base-Node***: MIB_BASE_ENTPRS or MIB_BASE_MGMT (see 4.1)
- define the ***MIB-Base-Node-Number***, which is the first single cipher following the MIB-Base-Node (e.g. '30060' for SEVENSTAX-MIB as an 'Enterprise-OID').
- create a GroupTable for this MIB (see 4.3 and 6.4.2)
- call the *stxMIB_RegisterModule()*.

All MIBs are save in a table of MIBs. While processing a SNMP-Request, the MIB-Module start to search inside the first registered MIB-Module. If the requested OID is not found or the requested OID is a lower number than the MIB contains, the search inside this MIB is stopped, and the next one in the list of registered MIBs is tried.

This results in a mix of sequential search through all registered MIBs and a tree-search inside the GroupTable and subsequent ObjectTable. It stops the search at the first tree-leaf, which is higher than the requested OID.

Please note, there is no need to register the MIB-Modules in the right (descending) order, because all MIBs get scanned. But it might speed up the search, if MIBs which get are often used are registered at first.

5 SNMP Module

5.1 Abstract

The SNMP Module is one of the both core elements (beside MIB) of SEVENSTAX-SNMP. The main tasks of this module are:

- processing incoming UDP packets
- executing SNMP protocol framework
- extracting message parameters (Message Type, Object Identifier)
- starting the MIB-Search process
- generate the adequate Response Message (based on MIB-Results)
- supporting Application Traps

The API distinguishes between the "***Agent***" and the "***Trap-Agent***": The "*Agent*" processes common SNMP tasks, while the "*Trap-Agent*" must separately be configured to enable transmission of SNMP-Traps.

Interaction between the SNMP-Module and the Application is reduced to a small API, which can be described with these steps:

1. Initializing and Setup of the SNMP-Agent and/or SNMP-Trap-Agent
2. Starting and Stopping the SNMP-Agent and/or SNMP-Trap-Agent

For details please see the public API below.

Any tasks concerning handling of Application Data, Application Managed Objects or registering Application MIB-Modules are processed by the separate application specific MIB-Module (e.g. see RFC1213-MIB or SEVENSTAX-MIB).

5.2 Data Types

The SNMP-Module does not need dedicated data types for its public API.

5.3 SNMP Functions

5.3.1 stxSNMP_Init (fnApplCallback, fnMIBHandler)

Description

Initializes SNMP, registers callbacks to application and MIB module, resets SNMP statistics and the the SNMP agent!

Parameter

PROTOCOL_NOTIFY_HANDLER fnApplCallback
Application NotifyHandler, gets called whenever SNMP has something essentials to noted to the user application.

MIB_PROCESS_FCT fnMIBHandler
MIB Handler function, establishes a runtime connection between SNMP and MIB

module. Should always be *stxMIB_Search()* (exchangeable for future work)

Return Value

BOOL_stx
TRUE_stx if anything works well, FALSE_stx else.

Comment

Does not start SNMP nor opens UDP ports.

The SNMP Agent is disabled by default. Please control the Agent by the Start/Stop API below.

The Application-CallbackHandler is currently not used but reserved for future.

5.3.2 stxSNMP_Tick ( )

Description

Checks for any pending SNMP request to be processed. Should be called as often as possible. Returns immediately if nothing to do.

Parameter

-

Return Value

-

Comment

-

5.3.3 stxSNMP_AgentSetup(usPort, szCommRd, szCommRdWr)

Description

Setups SNMP Agent parameters.

The SNMP Agent is responsible to correctly process incoming SNMP messages and to generate responses for the requesting SNMP Manager.

Parameter

UINT16_stx usPort
UDP Port number, on which the SNMP Agent should list on. (default: 161)

STRING_stx szCommRd
Reference (see Comment below) to Community String to be used for read access (default: “public”)

STRING_stx szCommRdWr
Reference (see Comment below) to Community String to be used for read/write access (default: “private”)

Return Value

BOOL_stx
TRUE_stx if anything works well, FALSE_stx on parameter error.

Comment

Please note, that the string parameters a saved as reference only! This means, that the given string pointers have to remain valid as long as SNMP works.

Does not start SNMP nor opens UDP ports.

5.3.4 stxSNMP_AgentStart ( )

Description

Starts the SNMP Agent and opens the UDP Agent port. Subsequent received SNMP messages will be processed.

Parameter

-

Return Value

BOOL_stx
TRUE_stx if Agent started and working, FALSE_stx if no free UDP port available or not yet initialized with valid parameters.

Comment

SNMP must be initialized with *stxSNMP_Init()* before.

Please note, that at least one MIB Module must be registered, before SNMP can process incoming SNMP requests (see *stxMIB_RegisterModule()* for details).

5.3.5 stxSNMP_AgentStop ( )

Description

Stops the SNMP Agent and closes the UDP Agent port. Subsequent received SNMP messages will no longer be processed.

Parameter

-

Return Value

-

Comment

-

5.3.6 stxSNMP_TrapAgentSetup (DestIP, usTrapPort, szTrapComm)

Description

Setups SNMP Trap Agent parameters.

The SNMP Trap Agent is responsible to initiate Trap messages and to send them to the prepared SNMP Manager.

Parameter

IPV4 DestIP

IP adress of the Manager to receive Traps.

UINT16_stx usTrapPort
UDP Port number, to which the SNMP Trap Agent will send Traps too (default: 162)

STRING_stx szTrapComm
Reference (see Comment below) to Trap Community String to be used when sending Traps (default: “private”)

Return Value

BOOL_stx
TRUE_stx if anything works well, FALSE_stx else.

Comment

Traps are disabled by default. Please control the Trap-Agent by the Start/Stop API below.

Please note, that the string parameters a saved as reference only! This means, that the given string pointers have to remain valid as long as SNMP works.

5.3.7 stxSNMP_TrapAgentStart ( )

Description

Starts the SNMP Trap-Agent and opens the UDP Trap port. Any subsequent call of *stxSNMP_TrapAgentSend()* will transmit a SNMP Trap Message.

Parameter

-

Return Value

BOOL_stx
TRUE_stx if Trap-Agent is started, FALSE_stx if no free UDP port available or not yet initialized with valid parameters.

Comment

The Trap-Agent must be initialized with *stxSNMP_TrapAgentSetup()* before.

5.3.8 stxSNMP_TrapAgentStop ( )

Description

Stops the SNMP Trap-Agent and closes the UDP Trap port. No SNMP Trap Message will be sent,

Parameter

-

Return Value

-

Comment

-

5.3.9 stxSNMP_TrapAgentSend (aulTrapOID)

Description

Immediately sends out a SNMPv2-TRAP Enterprise conform Trap message.

Uses the given OID, searches it inside the registered MIBs and sends OID & OID-Value out, together with SNMPv2-TRAP-OID and Timestamp..

Parameter

UINT32_stx FPTR_stx aulTrapOID
Reference to an array describing the OID as sequence of unsigned long words. The sequence must be null-terminated. The OID has to exist in one of the registered MIB modules.

Return Value

BOOL_stx
TRUE_stx if anything works well, FALSE_stx on parameter error, or OID not found or insufficient memory or UDP-Tx failed.

Comment

The Trap-Agent must be initialized and set up before sending a Trap Message.

5.4 Adjustable SNMP Parameters

The following parameters are pre-defined in 'snmp.h' with default values. Please do not change this file. As for all SEVENSTAX Internet Suite Protocols, the parameters can be set up for user requirements in '**features.h**', which will override the default settings.

5.4.1 SNMP Protocol Parameters

Change these parameters to adapt SNMP to your special SNMP-Agent behaviour:

SNMP_COMNAME_READONLY **default: "public"**

Defines a default value for "read-only" objects community name. Might be used for *stxSNMP_AgentSetup()*

SNMP_COMNAME_READWRITE **default: "private"**

Defines a default value for "read-write" objects community name. Might be used for *stxSNMP_AgentSetup()*

SNMP_PORT_PROT **default: 161** **valid:1..n**

UDP port used for Rx and Tx of SNMP messages

SNMP_PORT_TRAP **default: 162** **valid:1..n**

UDP port used for Rx and Tx of SNMP Trap messages

SNMP_GETBULK_MAX **default: 4** **valid:1..n**

Maximum number of objects in a response of a GET-BULK request.

5.4.2 SNMP Buffer Sizes

Please adapt the following values to your application to reduce internal memory consumption:

SNMP_COMMUNITYLEN_MAX **default: 50** **valid: 10..n**

Maximum size for the community string in bytes

SNMP_OIDDECLEN_MAX **default: 30** **valid: 8..n**

Maximum size for of a decoded ObjectID in Quad-Bytes.

SNMP_OIDENCLEN_MAX **default: 40** **valid: 8..n**

Maximum size for of a encoded ObjectID in Bytes.

SNMP_OBJDATA_MAX **default: 50** **valid: 8..n**

Maximum data size of one object a SNMP response in Bytes. Will be multiplied with SNMP_GETBULK_MAX objects in a response.

SNMP_SETVALUE_SIZE_MAX **default: 128** **valid: 8..255**

Maximum data size of one object a received SNMP SET request in Bytes.

SNMP_OIDSTRLEN_MAX **default: 40** **valid: 8..n**

Maximum size for of a encoded ObjectID string in Bytes (for debug printf() used only).

6 MIB Module

6.1 Abstract

The MIB Module is one of the both core elements (beside SNMP) of SEVENSTAX-SNMP. The main tasks of this module are:

- Registering user defined Management Information Bases (MIBs)
- Searching for Object Identifiers (OIDs) inside the MIBs
- Calling user specific GET/SET callback functions of the OIDs
- Delivering search results to the SNMP Module

The basic interface between SNMP and MIB is the call of *stxMIB_Search()*, which is internally done without need of application intervention.

Interaction between the MIB-Module and the User Application can be described with these steps:

1. Initializing and registering user specific MIBs
2. User/OID specific callback function for GET and SET

For details please see the public API below. Example code is given by the application level implementations of RFC1213-MIB and SEVENSTAX-MIB. Please take a closer look into these two modules on how to learn the preparation of customer specific MIBs.

Any task concerning handling of SNMP Protocol Data and Messages is processed by the separate SNMP-Module (see SNMP Module).

6.2 Internal Handling of OIDs

This chapter takes a closer look inside the OID handling of SEVENSTAX-SNMP, to help understanding the construction of customer defined MIBs.

Inside MIBs all branches and objects are identified by their Object-Identifier *OID*, which is a sequence of integer values. E.g. the RFC1213-MIB-Value '***sysDescr***' is uniquely defined by the OID '**.1.3.6.1.2.1.1.1.0**'.

Inside SEVENSTAX-SNMP an object is always identified with relative OIDs to their superior elements in these four steps: the MIB-Base-OID, the MIB-OID, Group-OID and the Object-OID. For the '*sysDescr*' above, this means:

a) the MIB-Base 'Management' is indicated by OID '**.1.3.6.1.2**'

b) the RFC1213-MIB-module is indicated by the succeeding OID '**.1**'

c) the GroupTable 'system' is identified by the OID '**.1**'

d) the Object 'sysDescr' is identified by the OID '**.1.0**'

This helps to reduce memory consumption, improves access time and minimizes effort to describe objects in Source code. Expressed in source, the steps above are supported by

a) a fixed Macro `SNMP_OID_ISO_DOD_INT_MGMT`

b) a call of the MIB-Registering function *stxMIB_RegisterModule()*

```
stxMIB_RegisterModule( SNMP_OID_ISO_DOD_INT_MGMT_MIB2,
       MIB_BASE_MGMT, MIB_1213_GroupTable, "RFC1213-MIB" );
```

c) a static *GroupTable* entry inside the *MIB_1213_GroupTable[ ]*

```
const MIB_GROUP_TYPE MIB_1213_GroupTable[] = {
```

supporting a callback function *stxMIB1213_GetGroup()* to return a group handler

```
{ 1, {1,0}, stxMIB1213_GetGroup, MIB_TABLE_STAT },
       // '.1.3.6.1.2.1.1.1' system
```

d) and a static *ObjectTable* entry inside *MIB_1213_Group_System[ ]*

```
const MIB_OBJECT_STAT_TYPE MIB_1213_Group_System[] = {
```

supporting the callback function *stxMIB1213_GetSystem()* to handle the object

```
{ 1, { 1,0}, SNMP_OBJT_OCTSTR, stxMIB1213_GetSystem, MIB1213_VARID_SYS_DESCR,
       SNMP_OIDS_READONLY }
```

For SEVENSTAX-SNMP these four steps are a mandatory structuring. But it is flexible too: The borderline to distinguish between a *GroupTable* and *ObjectTable* can be shifted and OIDs can be extended/reduced in length.

So parts of the Objects OID can e.g. be described in the common *GroupTable* identifier, to ease use of common handler functions. Or e.g. for an unique object (without common group members), most of the OID an be defined inside the *ObjectTable* instead of the *GroupTable*.

For details on how to describe MIBs please refer to chapter 4.3 Convert the MIB into Software, and the examples described in RFC1213-MIB and SEVENSTAX-MIB.

6.3 Static and Dynamic Objects

Some objects of a MIB are always defined and accessible. These objects have a fixed and always valid OID and are so called ***Static Objects***.

Other objects might be part of a table, which only exists under special runtime conditions, these are called ***Dynamic Objects***.

Please note: Only complete GroupTables can be defined as *Static* or *Dynamic*.

6.3.1 Static Objects and Tables

A list of static objects could be regarded as a table with a fixed dimension. GroupTables and Callback functions to handle such a predefined table can easily be arranged by the use of a common object callback handler function (e.g. see *stxMIB1213_GetSystem()*).

Static Tables are supported by SEVENSTAX-SNMP by a special GroupTable indicator *MIB_TABLE_STAT.*

6.3.2 Dynamic Objects and Tables

An example for this could be the list of current TCP/IP connections, which might be empty, if the device is not connected to the network. Another property of dynamic objects is - if they are part of n-dimensional array - that their OID itself consist of the n-dimension-table-index.

Dynamic Tables are supported by SEVENSTAX-SNMP too by a special GroupTable indicator *MIB_TABLE_DYN*. The structure of a *GroupTable* containing dynamic data is slightly different to a one containing static data: Only one set of index (incl. dimension) is defined. The decision, whether of and/or how many dynamic object(s) exist, is taken at runtime.

For details on how to create and use dynamic tables, please refer to MIB_OBJECT_DYN_TYPE and the examples prepared in RFC1213-MIB.

6.4 Data Types

Knowledge of the following data types is essential to create your own MIB. Please assure to understand chapter 6.2 “Internal Handling of OIDs“ before going on with this chapter.

6.4.1 MIBs, GroupTables and ObjectTables

The MIB-Module has an interface to the user application, which is one (or more) customer specific MIB module. SEVENSTAX-SNMP supports

- one (or more) *MIB-Modules*, which contain
- one (or more) *GroupTables*, which contain
- one (or more) *ObjectTables*, which contain
- one (or more) *Objects*

A *MIB-Module* is a list of *GroupTables* residing inside one of the registered MIBs.

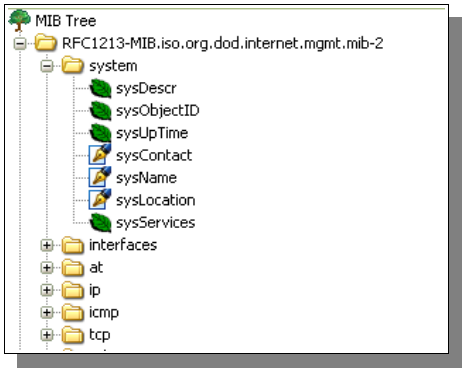
A *GroupTable* is a list several *ObjectTables* (defined by a common Branch-OID), which all are part of the same basic group (any main branch of the MIB).

A *ObjectTable* is a list several *Objects*, which all are part of the same group (branch/sub-branch of a MIB).

A *Object* is the atomic data structure, defined by a SNMP data type and it's ObjectIdentifier.

Example:

As an example, please see the RFC1213-MIB beside:



- '***RFC-1213-MIB***' is the *MIB-Module* containing several *GroupTables* like 'system' or 'interfaces'.
- '***system***' is one *GroupTable*, which contains several objects like 'system' or 'interfaces'.
- '***sysDescr***' is one object inside the *GroupTable*

The following chapters describe the structure of these modules, tables and objects. These types have to bes used to design a customer specific MIB.

Please note, that example usage of all these data types is shown inside the sample implementation for RFC1213-MIB and SEVENSTAX-MIB.

6.4.2 MIB_GROUP_TYPE

This is the basic element to construct a MIB. It defines one *GroupTable* as a part of a *MIB-Module*:

```
/* OID-Group type inside a Module table */
typedef struct _tag_MIB_GROUP_TYPE
{
    UINT8_stx       ucSize;                     /* size of the ulGroup field */
    UINT32_stx      ulOID[MIB_SUBOIDLEN_MAX];   /* Objectidentifier, MUST be closed with '0x0' */
    MIB_CBFCT_GROUP pGroupFn;                   /* fct pointer to access a object-group */
    UINT8_stx       ucTableType;                /* table type */
} MIB_GROUP_TYPE;
```

- ucSize defines the number of valid OID part numbers in the following array of OID parts.
- ulOID is a list of DWORD, representing each cipher inside the Sub-OID of this group
- pGroupFn is a callback function returning the reference to the adequate GroupTable
- ucTableType is an indicator, whether this group as a dynamic or static table.

For an example, please refer to "*Creating a GroupTable".*

6.4.3 MIB_OBJECT_STAT_TYPE

This structure describes a *Static Objects* as an entry of a *Static GroupTable* (see 6.3):

```
/* Object type inside STATIC MIB tables */
typedef struct
{
    UINT8_stx           ucSize;                     /* size of the ulOID field */
    UINT32_stx          ulOID[MIB_SUBOIDLEN_MAX];   /* Object identifier (decoded OID) */
    UINT8_stx           ucObjType;                  /* Object type */
    MIB_CBFCT_OBJSTAT   pObjStatFn;                 /* function pointer to Rd/Wr handler */
    UINT8_stx           ucVarID;                    /* Local Variable ID (may not equal Sub-OID) */
    UINT8_stx           ucRights;                   /* Access Rights */
} MIB_OBJECT_STAT_TYPE;
```

- ucSize defines the number of valid OID part numbers in the following array of OID parts
- ulOID is a list of dword numbers, representing each cipher inside the Sub-OID of this Object
- ucObjType indicates the SNMP data type (e.g. INT32)
- pObjStatFn is a callback function supporting GET/SET requests for this object
- ucVarID is a unique identifier to support a common handler function for different objects
- ucRights is an indicator for acces rights (read/readwrite) to the object

Example:

This is an example code of a *StaticGroupTable* containing several *StaticObject* entries:

```
/* System table 1.3.6.1.2.1.System.x */
STATIC_stx const MIB_OBJECT_STAT_TYPE MIB_1213_Group_System[] = {
{ 1,  { 1,0}, SNMP_OBJT_OCTSTR,  stxMIB1213_GetSystem, MIB1213_VARID_SYS_DESCR,  SNMP_OIDS_READONLY },
{ 1,  { 2,0}, SNMP_OBJT_OID,     stxMIB1213_GetSystem, MIB1213_VARID_SYS_OBJECTID, SNMP_OIDS_READONLY },
...
{ 1,  { 7,0}, SNMP_OBJT_INT32,   stxMIB1213_GetSystem, MIB1213_VARID_SYS_SERVICES, SNMP_OIDS_READONLY },
 MIB_OBJECT_STAT_LAST
};
```

In this example, *stxMIB1213_GetSystem()* is a common callback handler function, which gets called if the MIB searches for any of the Objects residing inside the '*system*' branch.

The callback function assumes a unique VariableID *MIB1213_VARID_SYS_xxx*, to distinguish between these objects. The callback accepts a write-acces, if the given user-right are setup in this GroupTable.

6.4.4 MIB_OBJECT_DYN_TYPE

This structure describes a *Dynamic Objects* as an entry of a *Dynamic GroupTable* (see 6.3):

```
/* Object type inside DYNAMIC MIB tables */
typedef struct _tag_MIB_OBJECT_DYN_TYPE
{
    UINT8_stx           ucSize;                         /* size of the ulOID field */
    UINT32_stx          ulOID[MIB_SUBOIDLEN_MAX];       /* Object identifier (decoded OID) */
    UINT8_stx           ucObjType;                      /* Object type */
    UINT8_stx           ucObjIdxType[MIB_OBJIDXTYPE_MAX];  /* Object Index Type */
    MIB_CBFCT_OBJDYN    pObjDynFn;                      /* function pointer to Rd/Wr object*/
    UINT8_stx           ucVarID;                        /* Local Variable ID (may not eq. Sub-OID) */
    UINT8_stx           ucRights;                       /* Access Rights */
} MIB_OBJECT_DYN_TYPE;
```

- ucSize defines the number of valid OID part numbers in the following array of OID parts
- ulOID is a list of dword numbers, representing each cipher inside the Sub-OID of this Object
- ucObjType indicates the SNMP data type (e.g. INT32)
- ucObjIdxType is a sequence of SNMP data types (e.g. INT32) of dynamic table index parts of the OID
- pObjDynFn is a callback function supporting GET/SET requests for this object
- ucVarID is a unique identifier to support a common handler function for different objects
- ucRights is an indicator for acces rights (read/readwrite) to the object

Example:

This is an example code of a *DynamicGroupTable* containing a *DynamicObject* entries:

```
/* dynamic UDP table - 1.3.6.1.2.1   .7.5 */
STATIC_stx const MIB_OBJECT_DYN_TYPE MIB_1213_Group_UDPTable[] = {
    // ucSize, ulOID, ucObjType,          ucObjIdxTypes,                      pObjDynFn,
                      ucVarId,                                    ucRights
    {  2,      {1,1}, SNMP_OBJT_IPADDR, { SNMP_OBJT_IPADDR, SNMP_OBJT_INT32 }, stxMIB1213_GetUPDTable,
                      MIB1213_VARID_UDP_TE_LOCALADDRESS, SNMP_OIDS_READONLY},
    {  2,      {1,2}, SNMP_OBJT_INT32,  { SNMP_OBJT_IPADDR, SNMP_OBJT_INT32 }, stxMIB1213_GetUPDTable,
                      MIB1213_VARID_UDP_TE_LOCALPORT,    SNMP_OIDS_READONLY},
    MIB_OBJECT_DYN_LAST
};
```

In this example, *stxMIB1213_GetUDPTable()* is a common callback handler function, which gets called if the MIB searches for any of element residing inside the dynamic 'UDP Connections Table'.

The callback function uses the unique VariableID to indicate which part of the UDP Table gets returned: MIB1213_VARID_UDP_TE_LOCALADDRESS_*xxx* distinguishes between these objects. The callback accepts a write-acces, if the given user-right are set up in this GroupTable.

The callback function additionally retrieves dynamic data extracted by the SNMP module, which indicate the index currently requested by the SNMP Manager (see Callback MIB_CBFCT_OBJDYN).

6.5 MIB Functions

6.5.1 stxMIB_Init ( )

Description

This function initializes the MIB Module. It Clears the list of registered MIBs. It should be called at Init-Time of all the other Protocol-Init-functions too.

Parameter

-

Return Value

-

Comment

Should be called first, before using any MIB-API function. Call *stxMIB_RegisterModule()* to register application specific MIBs afterwards.

6.5.2 stxMIB_Tick ( )

Description

This function keeps the MIB Module alive and should be called as often as possible.

Parameter

-

Return Value

-

Comment

The function is currently not used, but prepared for future use.

6.5.3 stxMIB_RegisterModule(ulMIBBaseOID, ucMIBType, pGroupTable, szMIBName)

Description

Registers a given user defined MIB module to the list of internal MIB modules.

After initialisation, the list of MIB-Modules is empty. To enable GET/SET access to the user defined OIDs of the MIB, the MIB has to be registered at runtime with this function.

Parameter

UINT32_stx ulMIBBaseOID
The Base OID of the given module (e.g. companies IBAN number)

UINT8_stx ucMIBType
Kind of module MIB type: MIB_BASE_MGMT / MIB_BASE_ENTPRS

MIB_GROUP_TYPE FPTR_stx pGroupTable
Reference to user defined const array of *MIB-GroupTable* for this MIB Module (see Comments below).

CHAR_stx FPTR_stx szMIBName
The MIB-Module name, just a debug helper.

Return Value

BOOL_stx
TRUE_stx if anything ok, FALSE_stx on parameter error

Comment

The *MIB-GroupTable* is the basic search element of the MIB module. Please refer to chapter 6.4.2, MIB_GROUP_TYPE on how to prepare a *GroupTable*.

6.5.4 Callback MIB_CBFCT_GROUP (ucTableNr)

Description

Returns the reference to the requested GroupTable (static/dynamic objects).

The function is given as one of the predefined elements inside a user defined MIB-Module (Group-Table of kind MIB_GROUP_TYPE). This callback function has to be implemented by the user application at least once for each registered MIB-Module.

Parameter

UINT8_stx ucTableNr
Sub-OID part of the MIB-OID (first cipher behind MIB-OID-part), to be used to select the adequate static/dynamic *GroupTable*

Return Value

void FPTR_stx pGroupTable
Reference to the adequate static/dynamic GroupTable, NULL_stx if not found.

Comment

When searching inside the MIB-Modules, this is the first user function which gets called by the SEVENSTAX-SNMP. It is responsible to return the reference to that GroupTable, which is able to support the delivered Sub-OID.

Please see the given *stxMIB1213_GetGroup() <TBD>* and *stxOID_GetTable() <TBD>* as examples on how this callback function works.

6.5.5 Callback MIB_CBFCT_OBJSTAT (Cmd,Var,ppdata,SetObjtT,VarObjT,Rights,pLen)

Description

Support the **GET** and/or **SET** methods for a requested ***Static Object.***

The function is given as one of the predefined elements inside a user defined GroupTable for each static object. This callback function has to be implemented by the user application at least once for each predefined GroupTable.

Parameter

UINT8_stx ucCommand
One of the SNMP command (GET/SET/...)

UINT8_stx ucVariable
Unique Variable-ID (might be the Sub-OID-part) inside the *GroupTable*

void FPTR_stx FPTR_stx ppData
Address of Reference (Ptr to Ptr!) to GET/SET object data.
If ucCommand is a SET-Req.: to be used to save the given data (if possible)
If ucCommand is a GET-Req.: to be used to return the reference to the requested data (if possible)

UINT8_stx ucSetObjectType
If ucCommand is a SET-Req.: The delivered SNMP-Object type (e.g. INT32),
If ucCommand is a GET-Req.: not used

UINT8_stx ucVarObjectType
Predefined Object Type of this object inside the local *GroupTable* (e.g. INT32).
If ucCommand is a SET-Req.: The object type is already proofed to be ok.
If ucCommand is a GET-Req.: Might be used for an additional type consistence check together with the *ucVariable* parameter.

UINT8_stx ucRights
Object access rights (read-write, read-only)
If ucCommand is a SET-Req.: Access rights are already proofed to be ok.
If ucCommand is a GET-Req.: Might be used for an additional check together with the *ucVariable* parameter.

UINT8_stx FPTR_stx pucLength
If ucCommand is a SET-Req.: Length of the delivered Object in bytes
If ucCommand is a GET-Req.: Length of the returned Object in bytes

Return Value

UINT8_stx SNMP_Error
Returns the adequate SNMP-Error, depending on the result of the requested SET/GET method (one of the SNMP_ERR_xxx codes).

Always assure to return valid values of *ppData* and *pucLength*, if returning SNMP_ERR_NOERROR!

Comment

If searching inside the MIB-Modules, this user function gets called by the SEVENSTAX-SNMP. It is responsible to execute the requested GET or SET method.

Commonly, this callback function has to be implemented once for each supported object. But the implementation can be simplified by using one common callback function and distinguishing the object by the help of the given *ucVariable* value.

Please see MIB_OBJECT_STAT_TYPE for details on how to define a static object inside a GroupTable of static objects.

Please see the the given *stxMIB1213_GetSystem() <TBD>* and *stxOID_FctManager() <TBD>* as examples on how this callback function works.

6.5.6 Callback MIB_CBFCT_OBJDYN (Cmd,Var,ppdata,RetIdx,SetObjtT,VarObjT,Rights,ObjTIdx,pulOID,pLen)

Description

Support the **GET only** method for a requested ***Dynamic Object***.

The function is given as one of the predefined elements inside a user defined GroupTable for each dynamic object. This callback function has to be implemented by the user application at least once for each predefined GroupTable.

Parameter

UINT8_stx ucCommand
One of the SNMP command (GET/SET/...)

UINT8_stx ucVariable
Unique Variable-ID (might be the Sub-OID-part) inside the *GroupTable*

void FPTR_stx FPTR_stx ppGetSetData
Address of Reference (Ptr to Ptr!) to GET/SET object data.
If ucCommand is a SET-Req.: to be used to save the given data (if possible)
If ucCommand is a GET-Req.: to be used to return the reference to the requested data (if possible)

UINT32_stx FPTR_stx FPTR_stx ppulReturnIdx
Address of Reference (Ptr to Ptr!) to returned Table Index of the requested GET object data.

UINT8_stx ucSetObjectType
If ucCommand is a SET-Req.: The delivered SNMP-Object type (e.g. INT32),
If ucCommand is a GET-Req.: not used

UINT8_stx ucVarObjectType
Predefined Object Type of this object inside the local *GroupTable* (e.g. INT32).
If ucCommand is a SET-Req.: The object type is already proofed to be ok.
If ucCommand is a GET-Req.: Might be used for an additional type consistence check together with the *ucVariable* parameter.

UINT8_stx ucRights
Object access rights (read-write, read-only)
If ucCommand is a SET-Req.: Access rights are already proofed to be ok.
If ucCommand is a GET-Req.: Might be used for an additional check together with the *ucVariable* parameter.

UINT8_stx FPTR_stx pucObjTypeIdx

Reference to list of object types inside prepared Dynamic GroupTable, might be helpful to create the necessary table index data objects preceding the requested object data

UINT8_stx FPTR_stx pulOID
Sub OID part containing the delivered indeces (if any)

UINT8_stx FPTR_stx pucLength
If ucCommand is a SET-Req.: Length of the delivered Object in bytes
If ucCommand is a GET-Req.: Length of the returned Object in bytes

Return Value

UINT8_stx SNMP-Error
Returns the adequate SNMP-Error, depending on the result of the requested GET method (one of the SNMP_ERR_xxx codes).

Always assure to return valid values of *ppData* and *pucLength*, if returning SNMP_ERR_NOERROR!

Comment

Please note, that SET-Requests inside dynamic GroupTables are not supported.

When searching inside the MIB-Modules, this user function gets called by the SEVENSTAX-SNMP. It is responsible to execute the requested GET method.

Please see MIB_OBJECT_DYN_TYPE for details on how to define a dynamic object inside a GroupTable of dynamic objects.

Please see the the given *stxMIB1213_GetARPTable() <TBD>* as an example on how this callback function works.

6.6 Adjustable MIB Parameters

The following parameters are pre-defined in 'mib.h' with default values. Please do not change this file. As for all SEVENSTAX Internet Suite Protocols, the parameters can be set up for user requirements in '**features.h**', which will override the default settings.

MIB_MODULE_MAX **default: 5** **valid: 1..n**
Maximum number of supported MIB modules added by *stxMIB_RegisterModule()*.

MIB_SUBOIDLEN_MAX **default: 3** **valid: ..n**
Maximum count of Sub-OIDs ciphers in Quadbytes in the GroupTable / ObjectTables. (see MIB_GROUP_TYPE)

MIB_OIDDECIDXLEN_MAX **default: 30** **valid: 1..n**
Maximum count of Sub-OIDs ciphers in (Quadbytes) to be used to search in dynamic tables. Must be adapted to the maximum application used index value inside OID-Tables (e.g. ARP-Cache).

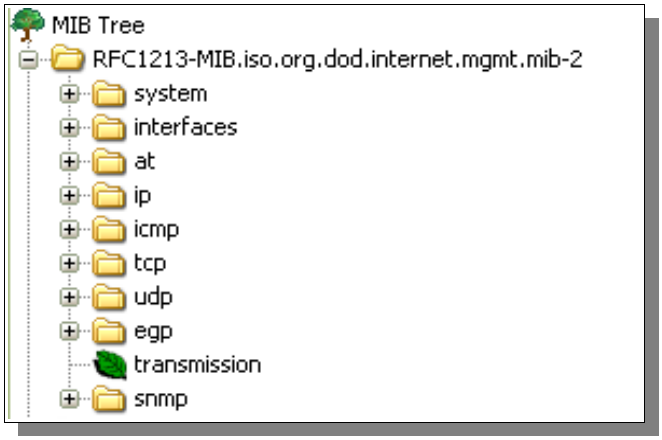
MIB_OIDENCIDX_MAX **default: 5** **valid: 1..n**
Maximum index count to be used if requesting dynamic objects.

7 RFC1213-MIB

This chapter describes an example MIB application. It is not mandatory to be used inside a customer application. It therefore can be switched of by defining MIB1213_SUPPORTED = 1 in 'features.h'.

7.1 Abstract

The RFC1213 describes a MIB especially created to manage information for TCP/IP based network devices. It contains a several number of structured information about the network interface, TCP and UDP settings, SNMP status and so on (see http://www.ietf.org/rfc/rfc1213.txt for details).



SEVENSTAX-SNMP includes a ready to use RFC1213-MIB example Implementation with a selection OIDs of these MIB Objects:

- system – System Information
- at – ARP table
- tcp – TCP connection table
- udp – UDP connection table
- snmp – some SNMP statistics

SEVENSTAX prepared these most essential RFC1213 objects, because they might be useful for a customer application too. Furthermore these samples (especially the dynamic tables) might be used as demonstration on how to use and create user defined MIBs.

Please contact SEVENSTAX, if you need a more extensive version of the RFC1213.

7.2 Using the RFC1213-MIB

SEVENSTAX prepared a ready to use SNMP Reference Application in '*ref_app_snmp.c*', in which RFC1213-MIB already is integrated. The functionality is implemented in the module '*mib_1213.c*'.

If you like to add RFC1213-MIB support to another project, please follow these steps:

- add the delivered source modules 'mib_1213.c/h' to your project
- enable *MIB1213_SUPPORTED =1* in 'features.h'
- Call *stxMIB_1213_Init()* once at application init time
- Call *stxMIB_1213_Tick()* as often as possible at application run time

7.3 Public Functions

The following functions are the public part of the RFC1213-MIB module.

Please note, that a special set of functions reside inside 'mib_1213.c', which directly request for information of the SEVENSTAX-TCP/IP suite modules (e.g. UDP-Table). These functions should not be altered by the user application.

7.3.1 stxMIB_1213_Init (fctApplCallback)

Description

This function initializes the RFC1213 MIB module. It calls the register function of the prepared RFC1213-MIB and save a dedicated callback handler to request user application 'system' data.

Parameter

MIB1213_CBFCT_APPL fctApplCallback
Reference to a callback function, which supports GET/SET of 'system' objects of the RFC1213-MIB.

Return Value

-

Comment

Should be called once at application startup. Please see MIB1213_CBFCT_APPL for details on how to create the callback function.

7.3.2 stxMIB_1213_Tick ( )

Description

This function provides the RFC1213-MIB with processing time. It should be called as often as possible.

Parameter

-

Return Value

-

Comment

Currently not used, but reserved for future tasks.

7.3.3 Callback MIB1213_CBFCT_APPL (ucVarID, ppData, pDataLen)

Description

Provides access to the application specific 'system' data of MIB1213-MIB. The RFC1213-MIB module will process GET and SET methods with this data.

This callback function has to be implemented by the user application. The reference to this function will be given as a parameter for *stxMIB_1213_Init ()*.

Parameter

UINT8_stx ucVarID
One of the MIB1213_VARID_SYS_xxx Variable IDs

void FPTR_stx FPTR_stx ppData

Address(!) of reference to requested data, used to GET or SET application data

UINT8_stx FPTR_stx pMaxLen
Reference to max buffer length in bytes for SET access.

Return Value

UINT8_stx buffer size
Real used buffer length for GET access. 0 if variable not supported/available.

Always assure to return valid values of *ppData* and *pucLength*, if returning a pMaxLen or return value != 0 !

Comment

-

7.4 Adjustable Parameters

The following parameters are pre-defined in 'mib.h' with default values. Please do not change this file. As for all SEVENSTAX Internet Suite Protocols, the parameters can be set up for user requirements in '**features.h**', which will override the default settings.

MIB1213_SUPPORTED **default: 1** **valid: 0..1**

Enables the support of the prepared RFC1213-MIB.

MIB1213_DEBUG **default: 1** **valid: 0..1**

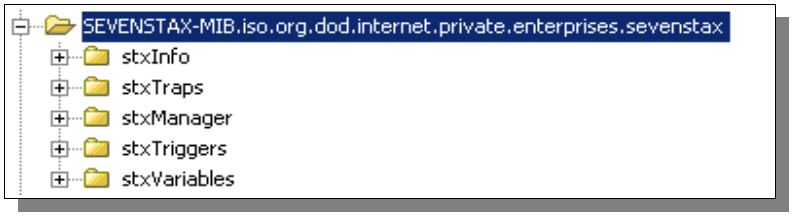
Enables debugouts for the RFC1213-MIB module.

8 SEVENSTAX-MIB

This chapter describes an example MIB application. It is not mandatory to be used inside a customer application. It therefore can be switched of by defining STXOID_SUPPORTED = 1 in 'features.h'.

8.1 Abstract

SEVENSTAX prepared this MIB as a showcase, on how to use all features of the SEVENSTAX-SNMP. It contains a number of predefined objects, including all supported data types and an example Trap application



SEVENSTAX uses the IANA-reserved Company-ID '30060' as part of the 'private' OID area. It demonstrate the following functionality:

- MIB specific version number - stxInfo
- Trap configuration for 4 independent traps - stxTraps
- SNMP-Manager configuration - stxManager
- a set of hardware support trigger objects - stxTrigger
- a set of demonstration objects with all supported MIB data types - stxVariables

For details of the SEVENSTAX-MIB structure and syntax, please see SEVENSTAX-MIB.

This MIB uses SEVENSTAX Reference Application features and therefore should be unaltered. But it might be used to analyse on how MIB creation and usage works. Please contact SEVENSTAX, if you need support on creating an customer specific MIB.

8.2 Using the SEVENSTAX-MIB

SEVENSTAX prepared a ready to use SNMP Reference Application in '*mib_sevenstax.c*', in which the complete support of the MIB description file 'SEVENSTAX-MIB' is integrated.

If you like to add SEVEBNSTAX-MIB support to another project, please follow these steps:

- add the delivered source modules 'sevenstax_mib.c/h' to your project
- enable *STXOID_SUPPORTED =1* in 'features.h'
- Call *stxOID_Init()* once at application init time
- Call *stxOID_TrapInit()* once at application init time
- Call *stxOID_Tick()* as often as possible at application run time

8.3 Generating Traps

SEVENSTAX prepared an universal setup for Traps, which enables a widely configurable usage of any Trap-Trigger and Trap-Information. The basic principle is, that any information to be used for Traps is accessible by SNMP itself – without user interaction. This means, that any information is already existent as an MIB-Object.

This is the way it works:

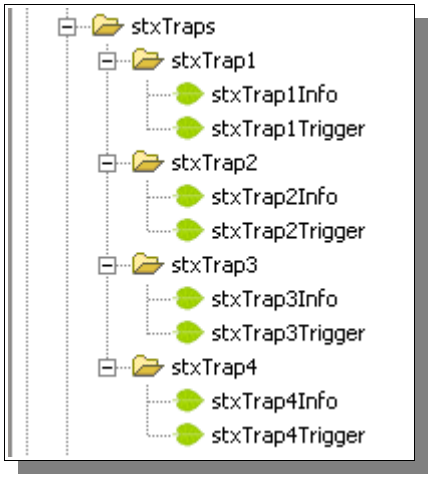
1. A configurable MIB-Object is used to trigger a Trap.
2. A configurable MIB-Object is used to be send as addtional information with a Trap.
3. A configurable SNMP-Manager will receive the Trap.

All values have a predefined value. They can be altered by access with a common MIB-Browser.

8.4 MIB Structure

The complete SEVENSTA-MIB can be found in chapter 10, SEVENSTAX-MIB.

8.4.1 stxTraps



There a 4 Traps available as MIB-Objects: **stxTrap1** ... **stxTrap4**. Each of these Traps consists of a Info-Object and Trigger-Object.

The MIB-Object referenced by the **stxTrap_Info** will be sent as an additional Infomation when sending the Trap.

The MIB-Object referenced by the **stxTrap_Trigger** will be used to trigger the Trap. It therefore should return a boolean information like 0/1.

Both are defined OIDs! This means, any OID (e.g. from another MIB-Module too) can be inserted here.

At runtime, the '*mib_sevenstax.c*' module internally and cyclically scans for the 4 Trigger Objects by requesting the MIB-value. If a Trigger Object returns a value != 0, the Trap gets triggered! Additionally, the MIB gets scanned for the Info-Object. Both are inserted into a Trap objects and get sent as a SNMPv2-Trap-Message.

The predefined Trap configuration is:

- Trap1: Trigger stxStateKey1, Info stxMIBVersion
- Trap2: Trigger stxStateKey1, Info stxtestSTRING
- Trap3: Trigger stxADCLimit, Info stxADCSample
- Trap4: unconfigured

8.4.2 stxTrapManager

Traps have to be sent to an available Trap-Manager, which will process the Trap and decide further action (e.g. requesting further parameters from the SNMP-Agent).

Therefore the SNMP-Agent has to know the Trap destination (the SNMP-Manager). The essential parameter can be set up in stxManager:

- the SNMP-Manager IP-Address
- the Trap-Reception UDP-Port (default 162)
- the Community-name to be used (default: private)

8.4.3 stxTriggers

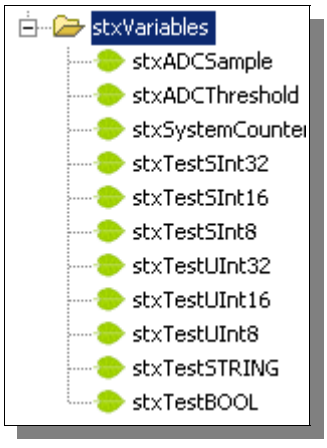
Traps have to be triggered by the value of MIB-Objects. Any MIB-Object returning integer values 0/1 can be used for stxTrap_Trigger. To simplify the configuration, SEVENSTAX prepared some boolean-like MIB-Objects in stxTriggers:

- **stxStateKey1** – returns 1 if hardware Key1 is pressed (0 else)
- **stxStateKey2** – returns 1 if hardware Key2 is pressed (0 else)
- **stxStateLED1** - returns 1 if hardware LED1 is ON (0 else)
- **stxStateADCLimit** - returns 1 if stxADCSample is bigger than stxADCLimit (0 else)

Of course, these values can be requested by common SNMP-GET requests too, e.g. by cyclically polling the value in a graph window.

8.4.4 stxVariables

An extended set of MIB-Objects have been prepared to show read/write access to any kind of supported MIB data type.



Name/OID	Value	Type
stxADCSample.0	500	Gauge
stxADCThreshold.0	500	Gauge
stxSystemCounter.0	2 hours 35 minutes 37 seconds	TimeTicks
stxTestSInt32.0	-323232	Integer
stxTestSInt16.0	-1616	Integer
stxTestSInt8.0	-8	Integer
stxTestUInt32.0	32000	Gauge
stxTestUInt16.0	1600	Gauge
stxTestUInt8.0	80	Gauge
stxTestSTRING.0	MIB Teststring	OctetString
stxTestBOOL.0	0	Integer

8.5 Public Functions

8.5.1 stxOID_Init ( )

Description

This function initializes the SEVENSTAX-MIB module.

It calls the register function of the prepared SEVENSTAX-MIB and saves a dedicated callback handler to request user application data.

Parameter

-

Return Value

-

Comment

Should be called once at application startup.

8.5.2 stxOID_TrapInit ( )

Description

This function initializes and starts the Trap-Testgenerator of the SEVENSTAX-MIB module.

It calls the *stxSNMP_TrapAgentSetup()* of the prepared SEVENSTAX-MIB and saves default values for the Trap configuration as shown in 8.3, Generating Traps.

Parameter

-

Return Value

-

Comment

If called, a cyclically check of all Trap-Trigger conditions is executed every

8.5.3 stxOID_Tick ( )

Description

This function provides the SEVENSTAX-MIB module with processing time. It should be called as often as possible.

Parameter

-

Return Value

-

Comment

Currently not used, but reserved for future tasks.

8.6 Adjustable Parameters

The following parameters are pre-defined in 'mib.h' with default values. Please do not change this file. As for all SEVENSTAX Internet Suite Protocols, the parameters can be set up for user requirements in '**features.h**', which will override the default settings.

STXOID_SUPPORTED **default: 1** **valid: 0..1**

Enables the support of the prepared SEVENSTAX-MIB.

STXOID_DEBUG **default: 1** **valid: 0..1**

Enables debugouts for the SEVENSTAX-MIB module.

STXOID_TRAPDEMO **default: 1** **valid: 0..1**

Enables Trap-Checking and Generating part of the SEVENSTAX-MIB module.

STXOID_MANAGER_IP **default:** **{ 192, 168, 130, 24 }**

Sets up the default value for SNMP manager to receive the Trap messages.

STXOID_MANAGER_PORT **default: 162**

Sets up the default value for SNMP manager UDP port to receive the Trap.

STXOID_TRAP_CHECKCYCLE **default: 250** **valid: 250..1000**

Defines Trap-Checking cycle time in ms.

STXOID_MANAGER_COMM **default:** **'private'**

Defines the default value of the SNMP community used when sending Traps.

STXOID_TRAPDEBUG **default: 1** **valid: 0..1**

Enables debugouts for the Trap-Checking and Generating part of the SEVENSTAX-MIB module. (1 = generates lot of runtime debugouts!)

9 Tools

9.1 MIB-Validators

While designing a customer MIB, the file structure and syntax has to validated. This is essential to allow tools (like MIB Browsers or network analysers like WireShark) to import the designed MIB without errors.

- A starting point for MIB validating might be this IETF Website: http://tools.ietf.org/inventory/verif-tools.shtml
- As an example for several online MIB Validators you might look to here:http://wwwsnmp.cs.utwente.nl/ietf/mibs/validate/

9.2 MIB Code-Generators

SEVENSTAX-MIB uses an internal presentation of MIBs and object handlers, which is product specific. The conversion from a MIB-File into sourcecode has to be done manually, because there currently is no tool available, to auto-convert a given MIB into SEVENSTAX-SNMP compatible source code.

9.3 MIB Browsers

MIB-Browsers are standard tools mostly working as a SNMP-Manager, which is able to connect a SNMP-Agent. A large number of MIB-Browser are available (with costs or free) are available.

Most of these tools are able to scan for SNMP Traps too, or are able to visualize dynamic data by permanent polling of OIDs. This is a short overview of free MIB-Browsers:

- Comfortable and rich featured MIB-Browser with Trap-Receiver http://ireasoning.com/mibbrowser.shtml
- Rich featured MIB-Browser with Trap-Receiver and Graphical View http://www.manageengine.com/products/mibbrowser-free-tool/
- Simple SNMP-Tester for single Tests http://www.paessler.com/tools/snmptester
- Comfortable Network Device Monitor using SNMP http://www.solarwinds.com/products/freetools/network_device_monitor/

9.4 SNMP Simulator

- Simple SNMP Device Monitor for test purpose http://www.solarwinds.com/products/freetools/network_device_monitor/

10 Appendix A: SEVENSTAX-MIB

```
SEVENSTAX-MIB DEFINITIONS ::= BEGIN
IMPORTS
        enterprises, IpAddress, Gauge, TimeTicks
            FROM RFC1155-SMI
        OBJECT-TYPE
            FROM RFC-1212;

--  This MIB module uses the extended OBJECT-TYPE macro as
--  defined in [14];

--  $RCSfile: sevenstax-MIB,v $
--  $Revision: 1.13 $
--  $Date: 2010/06/15 15:00:35 $

-------------------------------------------------------------------------------
--  sevenstax Company
sevenstax     OBJECT IDENTIFIER ::= { enterprises 30060 }
-------------------------------------------------------------------------------
-- textual conventions

DisplayString ::=
    OCTET STRING
-- This data type is used to model textual information taken
-- from the NVT ASCII character set.  By convention, objects
-- with this syntax are declared as having
--
--      SIZE (0..255)

PhysAddress ::=
    OCTET STRING
-- This data type is used to model media addresses.  For many
-- types of media, this will be in a binary representation.
-- For example, an ethernet address would be represented as
-- a string of 6 octets.



-------------------------------------------------------------------------------
-- the sevenstax MIB Groups
-------------------------------------------------------------------------------

stxInfo         OBJECT IDENTIFIER ::= { sevenstax 1 }
stxTraps        OBJECT IDENTIFIER ::= { sevenstax 2 }
stxManager      OBJECT IDENTIFIER ::= { sevenstax 3 }
stxTriggers     OBJECT IDENTIFIER ::= { sevenstax 4 }
stxVariables    OBJECT IDENTIFIER ::= { sevenstax 5 }




-------------------------------------------------------------------------------
-- Common sevensatx MIB information
-------------------------------------------------------------------------------

stxMIBVersion OBJECT-TYPE
```

```
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Indicates the SEVENSTAX MIB Version,
                (Main/Subversion), which will be
                incremented on every structural
                change."
        ::= { stxInfo 1 }


-------------------------------------------------------------------------------
-- Definitions for Trap-Trigger and Trap-Information
-------------------------------------------------------------------------------

stxTrap1 OBJECT IDENTIFIER ::= { stxTraps 1 }

    stxTrap1Info OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
               "The object referenced by this OID
               will be sent, if this Trap occures.
               You can use any available OID."
        ::= { stxTrap1 1 }

    stxTrap1Trigger  OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "If the object referenced by this
                OID changes from '0' to '1', this
                Trap gets triggered. You can use
                e.g. one of the stxTriggers events.
                If the this OID is '0', this Trap
                is disabled."
        ::= { stxTrap1 2 }

stxTrap2 OBJECT IDENTIFIER ::= { stxTraps 2 }

    stxTrap2Info      OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
               "The object referenced by this OID
               will be sent, if this Trap occures.
               You can use any available OID."
        ::= { stxTrap2 1 }

    stxTrap2Trigger OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "If the object referenced by this
                OID changes from '0' to '1', this
                Trap gets triggered. You can use
                e.g. one of the stxTriggers events.
                If the this OID is '0', this Trap
```

```
                is disabled."
        ::= { stxTrap2 2 }

stxTrap3 OBJECT IDENTIFIER ::= { stxTraps 3 }

    stxTrap3Info  OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
               "The object referenced by this OID
               will be sent, if this Trap occures.
               You can use any available OID."
        ::= { stxTrap3 1 }

    stxTrap3Trigger OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "If the object referenced by this
                OID changes from '0' to '1', this
                Trap gets triggered. You can use
                e.g. one of the stxTriggers events.
                If the this OID is '0', this Trap
                is disabled."
        ::= { stxTrap3 2 }

stxTrap4 OBJECT IDENTIFIER ::= { stxTraps 4 }

    stxTrap4Info  OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
               "The object referenced by this OID
               will be sent, if this Trap occures.
               You can use any available OID."
        ::= { stxTrap4 1 }

    stxTrap4Trigger OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "If the object referenced by this
                OID changes from '0' to '1', this
                Trap gets triggered. You can use
                e.g. one of the stxTriggers events.
                If the this OID is '0', this Trap
                is disabled."
        ::= { stxTrap4 2 }


-------------------------------------------------------------------------------
 -- Definitions for the Remote SNMP Manager to receive the Traps
 -------------------------------------------------------------------------------

stxManagerIP OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-write
    STATUS  mandatory
```

```
    DESCRIPTION
            "The IP-Address of the SNMP Manager
            to receive a trap."
    ::= { stxManager  1 }

stxManagerPort OBJECT-TYPE
    SYNTAX  INTEGER (0..65535)
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The UDP-Port of the SNMP Manager
            to receive a trap."
    ::= { stxManager  2 }

stxManagerCommName OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The Community-Name to be used
            when sending a trap."
    ::= { stxManager  3 }


-------------------------------------------------------------------------------
-- prepared boolean Trap Trigger Variables
-------------------------------------------------------------------------------

stxStateKey1 OBJECT-TYPE
    SYNTAX  INTEGER ( 0..1 )
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "Boolean state of the hardware element.
             0 = Key1 not pressed
             1 = Key1 pressed
             Can be used to trigger Traps."
    ::= { stxTriggers 1 }


stxStateKey2 OBJECT-TYPE
    SYNTAX  INTEGER ( 0..1 )
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "Boolean state of the hardware element.
             0 = Key1 not pressed
             1 = Key1 pressed
             Can be used to trigger Traps."
    ::= { stxTriggers 2 }


stxStateLED1 OBJECT-TYPE
    SYNTAX  INTEGER ( 0..1 )
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "Boolean state of the hardware element.
             0 = LED1 off
             1 = LED1 on
             Can be used to trigger Traps."
    ::= { stxTriggers 3 }
```

```
stxStateADCLimit OBJECT-TYPE
    SYNTAX  INTEGER ( 0..1 )
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "Logical state, as result of
            comparison of 'stxADCSample'
            and 'stxADCThreshold':
            Sample <  Threshold: Limit = 0
            Sample >= Threshold: Limit = 1
            Can be used to trigger Traps."
    ::= { stxTriggers 4 }


-------------------------------------------------------------------------------
-- some sevenstax demonstration variables
-------------------------------------------------------------------------------

stxADCSample OBJECT-TYPE
    SYNTAX  Gauge
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "Current AD-Converter Value,
            if available on hardware."
    ::= { stxVariables 1 }

stxADCThreshold OBJECT-TYPE
    SYNTAX  Gauge
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "AD-Converter threshold,
            if available on hardware.
            Can be used to setup the
            'stxStateADCLimit' trigger.
            Sample <  Threshold: Limit = 0
            Sample >= Threshold: Limit = 1"
    ::= { stxVariables 2 }

stxSystemCounter OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "Current SystemTime in
            Milliseconds since last
            PowerUp. Can be used as
            'Info' for a Trap."
    ::= { stxVariables 3 }

stxTestSInt32 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "For SET/GET requests tests.
            Mapped to a 32-bit signed
            variable on target."
    ::= { stxVariables 4 }
```

```
stxTestSInt16 OBJECT-TYPE
    SYNTAX  INTEGER ( '7FFF'h .. '8000'h )
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "For SET/GET requests tests.
            Mapped to a 16-bit signed
            variable on target."
    ::= { stxVariables 5 }

stxTestSInt8 OBJECT-TYPE
    SYNTAX  INTEGER ( '7F'h .. '80'h )
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "For SET/GET requests tests.
            Mapped to a 8-bit signed
            variable on target."
    ::= { stxVariables 6 }

stxTestUInt32 OBJECT-TYPE
    SYNTAX  Unsigned32 (0..'ffffffff'h)
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "For SET/GET requests tests.
            Mapped to a 32-bit unsigned
            variable on target."
    ::= { stxVariables 7 }

stxTestUInt16 OBJECT-TYPE
    SYNTAX  Unsigned32 (0..'ffff'h)
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "For SET/GET requests tests.
            Mapped to a 16-bit unsigned
            variable on target."
    ::= { stxVariables 8 }

stxTestUInt8 OBJECT-TYPE
    SYNTAX  Unsigned32 (0..'ff'h)
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "For SET/GET requests tests.
            Mapped to a 8-bit unsigned
            variable on target."
    ::= { stxVariables 9 }

stxTestSTRING OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "For SET/GET requests tests.
            Mapped to a STRING buffer
            on target."
    ::= { stxVariables 10 }

stxTestBOOL OBJECT-TYPE
```

```
      SYNTAX  INTEGER ( 0..1 )
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
              "For SET/GET requests tests.
              Mapped to a BOOL-variable
              on target."
      ::= { stxVariables 11 }


--------------------------------------------------------------------------------
END
--------------------------------------------------------------------------------
```

11 Change History

Ver.	*Date*	*by*	*Change description*
1.0	31-01-2008	krz	base version
1.1	14-08-2009	bme	reviewed based on new implementation
1.2	15-10-2009	bme	reviewed based on product update, extended API
1.3	27-10-2009	krz	Review based on Code Review
1.4	19-08-2010	krz	Some API changes concerning RFC1213
1.5	22-10-2010	krz	Trap API simplified
1.6	19-01-2011	krz	Updated to new Product version V6.1
1.7	24-01-2011	krz	reviewed

The information furnished in this document is believed to be accurate and reliable. However, no responsibility is assumed by sevenstax for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of sevenstax.

This document is an intellectual property of sevenstax GmbH. Unauthorized copying and distribution is prohibited.

Copyright (c) 2011 by SEVENSTAX GmbH